WHAT IS CLAIMED IS:

1. A method for producing, for a target computer architecture and a program fragment, a near-optimal code sequence for executing the program fragment on the target computer, comprising:

repeatedly invoking an automatic theorem prover to

determine a minimum cycle budget that is the lowest of any cycle budget K for which a formalized mathematical conjecture that no code sequence for the target computer architecture executes the program fragment within the cycle budget K is unprovable by the automatic theorem prover, and

extract the near optimal code sequence from a counterexample implicit in the failed proof of the formalized mathematical conjecture for the minimum cycle budget.

2. The method of claim 1, wherein the automatic theorem prover is two-phased, the two phases being

a matcher for instantiating facts about machine operations that are computable by a machine with the target computer architecture and facts about non-machine operations, followed by

a boolean satisfiability search.

3. The method of claim 1, wherein the program fragment specifies a vector of expressions to be computed together with one or more of

a vector of target destinations into which the values of the expressions are to be placed, and

a guard and label pair, the guard being a given boolean expression that determines whether the program fragment is to be executed as described or whether, instead, control is to be transferred to the label.

4. The method of claim 1, wherein, during the invocations of the automatic theorem prover, the minimum number of machine cycles for each successive invocation is set to a value so as to bisect the interval of remaining possible values of the minimum number of machine cycles.

5. The method of claim 2, wherein the instantiated facts from the matcher are asserted into an e-graph which is formed from a term graph augmented by an equivalent relation connecting terms known to be equal.

6. The method of claim 2, wherein the satisfiability search operates on a collection of boolean unknowns that encode a set of conjectured code sequences for a machine with the target computer architecture, each of these code sequences being defined in terms of a set of machine operations initiated in each cycle.

7. The method of claim 6, wherein the instantiated facts from the matcher are asserted into an e-graph which is formed from a term graph augmented by an equivalent relation connecting terms known to be equal, and wherein the encoding is performed such that, for each term of the e-graph and each cycle i of the minimum number of machine cycles for a particular invocation, there is a particular boolean unknown that indicates whether the conjectured code sequence performs a computation of the root operation of the term during cycle i.

8. The method of claim 6, wherein the boolean unknown can encode alternative boolean constraints suitable for the target architecture.

9. A method for producing, for a target computer architecture and a program fragment, a near-optimal code sequence for executing the program fragment on the target computer, comprising:

repeatedly invoking an automatic theorem prover to prove unsatisfiable a formalized mathematical conjecture that, for a cycle budget K, no code sequence for the target computer architecture executes the program fragment within that cycle budget K,

wherein if the proof fails, a K-cycled program computing the program fragment is embedded in the failed proof,

wherein the near-optimal code sequence is found, and the invocation need not be repeated, when it is established that both the K-cycled program computes the program fragment and a cycle budget K-1 is insufficient in that the cycle budget K is minimum, the K-cycled program being extracted as the near-optimal code sequence, and

wherein, if the near-optimal code sequence is not found in a present invocation, for a next revocation of the automatic theorem prover if the proof succeeds the budget cycle K is doubled (K:=K*2) and if the proof fails the budget cycle is bisected (K:=K/2) and a new K-cycled program computing the program fragment that is embedded in the failed proof is extracted.

10. The method of claim 9, wherein the program fragment is presented to the automatic theorem prover as a set of guarded multi-assignments each including a guard and a multi-assignment that can be performed only when its respective guard is true.

11. The method of claim 10, wherein the set of guarded multi-assignments is compiled by instantiating universal facts about relevant and useful operators including machine and non-machine terms, wherein each instance of relevant and useful operators provides a way for computing a corresponding multi-assignment.

12. The method of claim 11, wherein the ways for computing the multi-assignments are encoded in a graph.

13. The method of claim 12, wherein the graph is an equivalence graph (e-graph) formed as a directed acyclic graph.

14. The method of claim 11, wherein the graph is transformed in the presence of equalities between nodes.

15. The method of claim 11, wherein the graph is submitted for the extraction of the near optimal code sequence, the extraction using, in addition, a description of the target computer architecture for formulating a boolean satisfiability problem a solution of which is found for the minimum cycle budget K via a satisfiability search.

16. The method of claim 12, wherein for a multi-assignment of the size n, an e-graph with a size order of n represents 2^n distinct ways of computing the multi-assignment.

17. The method of claim 9, wherein the extraction of the near optimal code sequence is done from a formulation of a boolean staisfiability problem using a set of boolean unknowns that are one-to-one corresponding to a solution of the boolean satisfiability problem, the solution corresponding to a budget-cycle machine program where the budget is the minimum cycle budget K.

18. A method for producing a near-optimal code sequence to be executed on a target computer, comprising:

repeatedly invoking an automatic theorem prover having a matcher and a solver, the automatic theorem prover being invoked to

perform, via the matcher, a goal-directed search for any expression specified in its declarative input language, wherein a successful proof corresponds to an unsuccessful search and vice-versa; and

perform, via the solver, a satisfiability search, based on results of the goal-directed search, that targets a cycle budget limited goal-oriented code sequence,

wherein the goal-directed search via the matcher produces a number of possible plans for creating the near-optimal code sequence, and wherein the satisfiability search selects an optimal plan from among the possible plans discovered via the matcher, thereby automatically producing the near-optimal code sequence.

19. The method of claim 18 wherein by way of the goal-directed search the matcher instantiates facts about machine operations that are computable by a machine with the target computer architecture and facts about non-machine operations.

20. The method of claim 1 wherein the automatic theorem prover performs refutation-based automatic theorem proving.

21. The method of claim 9 wherein the automatic theorem prover performs refutation-based automatic theorem proving.

22. A method for automatic generation of a near-optimal code sequence for execution on a computer, comprising:

repeatedly applying automatic theorem-proving to a code sequence generator, including

introducing a multi-assignment to the code sequence generator,

performing matching, and

performing planning with a satisfiability search,

wherein the matching produces a number of possible plans for creating the near-optimal code sequence, and wherein the planning selects an optimal plan from among the possible plans discovered via the matching, thereby automatically producing the near-optimal code sequence.

23. A method as in claim 22, wherein the multi-assignment includes goal terms that specify what result the near-optimal code sequence is expected to produce, and wherein the applying automatic theorem proving further includes initializing a term graph with the goal terms whereby nodes of the term graph receive the goal terms.

24. A method as in claim 23, wherein the matching includes

introducing instances of universal facts that are relevant to the near-optimal code sequence, and

augmenting the term graph with equivalence relations between the goal terms and corresponding instances of the universal facts by matching the universal facts against the term graph.

25. A method as in claim 22, wherein values of the goal terms are computed into registers of the computer, the registers being specified in the multi-assignment.

26. A method as in claim 24, wherein the term graph is augmented by the equivalence relations on its nodes to produce an equivalence graph (e-graph).

27. A method as in claim 26, wherein the matching transforms the e-graph into a transformed e-graph that is then provided to the planning for the satisfiability search.

28. A method as in claim 24, wherein the satisfiability search produces the near-optimal code sequence for achieving values corresponding to the goal terms.

29. A method as in claim 28, wherein the near-optimal code sequence is created from the term graph by iteratively solving a satisfiability problem with various cycle budgets until an optimal code sequence is found.

30. A method as in claim 24, wherein the universal facts are available in a file and are introduced as an input to the code sequence generator so that the universal facts can be changed without changing the code sequence generator.

31. A method as in claim 24, wherein the universal facts express properties of operators in the goal terms.

32. A method as in claim 25, wherein the term graph is initialized with node terms representing the goal terms.

33. A method as in claim 23, wherein the near optimal code sequence is targeted for producing the result within a cycle budget.

34. A method as in claim 33, wherein the cycle budget is a minimal number of cycles.

35. A method as in claim 22, wherein the satisfiability search is a goal-directed search.

36. A code sequence generation tool for automatic generation of a near-optimal code sequence, comprising:

an input capable of receiving a multi-assignment;

a matcher responsive to the multi-assignment and producing via matching a number of possible plans for creating the near-optimal code sequence; and

a planner configured to select via a satisfiability search an optimal plan from among the possible plans discovered by the matcher,

wherein the code sequence generation tool is configured to invoke the matcher and the planner thereby implementing an automatic theorem-proving for automatically generating the near-optimal code sequence.

37. A code sequence generation tool as in claim 36 being further configured for producing the optimal code sequence using a goal-oriented, cycle budget limited code sequence in generating the near-optimal code sequence.

38. A code sequence generation tool as in claim 36 wherein the planner includes a constraint generator and a solver, the code sequence generation tool further comprising an input configured for introducing architectural constraints to the constraint generator which the constraint generator uses in creating a set of boolean unknowns for the solver.

39. A code sequence generation tool for automatic generation of a near-optimal code sequence, comprising:

an input capable of receiving a multi-assignment;

matching means responsive to the multi-assignment and producing via matching a number of possible plans for creating the near-optimal code sequence; and

planning means configured to select via a satisfiability search an optimal plan from among the possible plans discovered by the matching means,

wherein the code sequence generation tool is configured to invoke the matching means and the planning means thereby implementing an automatic theorem-proving for automatically generating the near-optimal code sequence.

40. A computer readable medium embodying computer program code configured to cause a computer to performs steps for automatically generate a near-optimal code sequence, comprising:

applying automatic theorem-proving to a code sequence generator, including

introducing a multi-assignment to the code sequence generator,

performing matching, and

performing planning with a satisfiability search,

wherein the matching produces a number of possible plans for creating the near-optimal code sequence, and wherein the planning selects an optimal plan from among the possible plans discovered via the matching, thereby automatically producing the near-optimal code sequence.